LAUREA AMMATTIKORKEAKOULU Uuden edellä

Javan soveltuvuus DSiP-reititysjärjestelmän asetusmuokkaimen luontiin

Vilpas, Mika

2012 Leppävaara

Laurea-ammattikorkeakoulu
Leppävaara

Javan soveltuvuus DSiP-reititysjärjestelmän asetusmuokkaimen luontiin

Mika Vilpas
Tietojenkäsittely
Opinnäytetyö
Maaliskuu, 2012

Laurea-ammattikorkeakoulu **Tiivistelmä**
Leppävaara
Tietojenkäsittely

Mika Vilpas

Javan soveltuvuus DSiP-reititysjärjestelmän asetusmuokkaimen luontiin

Vuosi 2012 Sivumäärä 38

Tutkimuksen tavoitteena on selvittää, miten hyvin Java-ohjelmointikielen tarjoaman ominaisuudet soveltuvat tyypillisen graafisen ohjelman rakentamiseen. Koska Java on laajalti yritysmaailmassa käytetty ohjelmointikieli, monet yritykset saattavat valita sen ohjelmistokehityksen kieleksi yksinomaan tämän takia. Mikäli löytyisi kieli, joka soveltuisi yrityksen käyttöön, ja joka olisi Javaa tehokkaampi, yritys saisi tästä kilpailuedun kilpailijoihinsa nähden.

Tutkimus tehtiin rakentamalla DSiP (Distributed Systems Intercommunication Protocol) -monikanavareititysjärjestelmän asetustiedostoja hallitseva Java-ohjelma, sekä vertailemalla kehityksessä havaittuja kielen ominaisuuksia ja puutteita muihin ohjelmointikieliin. Kielen ominaisuuksien vertailussa otettiin huomioon eri ohjelmoinnin viitekehyksiä sekä suoritusympäristöjä.

Tutkimuksen tulosten perusteella Java on hyvä, mutta rajoittunut ohjelmointikieli. Se tarjoaa vankan pohjan ohjelmoinnin perustyökaluja, mutta sen kehitys on paikoitellen kankeaa ja työlästä. Javaan verrattuna on tarjolla monipuolisempia ohjelmointikieliä, jotka soveltuvat hyvin jopa samaan viitekehykseen ja suoritusympäristöön kuin Java itse.

Asiasanat: Java, ohjelmistokehitys, ohjelmointi, DSiP.

Laurea University of Applied Sciences **Abstract**
Laurea Leppävaara
Bachelor's Degree in Business Information Technology

Mika Vilpas

The suitability of Java in the creation of a DSiP routing system configuration editor

Year 2012 Pages 38

The purpose of this thesis' research is to discover how well the Java programming language is applied to building a typical graphical user interface program. Since Java is a widely used programming language in the software business, many organizations may choose it as their primary programming language solely for this reason. If there were to a language more effective than Java and suitable for a particular organization, it would gain a competitive advantage over its competitors.

The study was conducted by building a graphical user interface program that manages configuration files for a DSiP (Distributed Systems Intercommunication Protocol) multichannel routing system. In addition to building the program, the pros and cons of the language observed during program development were compared to other programming languages. The comparison took multiple programming paradigms and execution environments into account.

The research showed that Java is a good but limited programming language. It provides a solid set of basic programming tools, but its development, at times, is stiff and laborious. More versatile languages than Java are available to the programmers today, and some of these languages even apply to the same paradigm and execution environment as Java.

Keywords: Java, software development, programming, DSiP.

Sisällys

1 Johdanto

Suuri osa 2000-luvun läntisen maailman työnteosta on muuttunut tietotyöksi. Tietotyön tarkoitus on haalia ja jalostaa tietoa edelleen hyödyllisemmässä muodossa. Tieto itsessään voi olla eri muodoissa, mutta on hyvin yleistä että sitä säilytetään digitaalisessa muodossa. Digitaalisen tiedon käsittelyyn käytetään siihen soveltuvia tietokoneita.

Digitaalisen tiedon käsittelyyn, monistukseen ja siirtämiseen on olemassa kehittyneitä työkaluja, ja näiden työkalujen kehitystä pidetään tärkeänä. Tietokoneille annetaan tiedon käsittelyyn yksityiskohtaiset ohjeet tavallisimmin käyttäen ohjelmointikieliä. Näitä kieliä on nykypäivänä olemassa valtava määrä. Kielet eivät ole samanlaisia, vaan ne on suunniteltu ratkaisemaan eri ongelmia eri tavoilla.

Yksi suosituimmista ohjelmointikielistä on Java-niminen ohjelmointikieli. Java on laajasti käytössä yritysmaailmassa, ja sitä pidetään vakaana valintana yrityksen tuotekehityksen ohjelmointikieleksi.

Tämän opinnäytetyön tarkoituksena on selvittää Java-ohjelmointikielen soveltuvuutta DSiP-monikanavareititysjärjestelmän asetustiedostojen graafisen hallintaohjelman kirjoittamiseen. Yrityksille ja yksittäisille ohjelmoijille on tärkeää käyttää tehokkaita työkaluja ja ohjelmointikieliä ohjelmistojen rakentamiseen. Tämä on yrityksille selvä kilpailuetu, koska yritys kykenee tuottamaan saman määrän ominaisuuksia lyhyemmässä ajassa kuin kilpailijansa.

Tutkimuksen yhteydessä kehitettävän ohjelman tarkoitus on ratkaista hyvin yleinen ongelma. Sen tarkoitus on tarjota käyttäjälle yhtenäinen ja helposti hallittava näkymä koko järjestelmän toimintaan. Ohjelma vähentää käyttäjän päänvaivaa ja työmäärää. Mikäli käyttäjä ei käyttäisi ohjelmaa, hän joutuisi suorittamaan monimutkaisia tai työläitä toimenpiteitä itse.

Tutkimus koostuu vikasietoiseen DSiP (Distributed Systems Intercommunication Protocol) monikanavareititysjärjestelmään suunnatun ohjelman rakentamisesta sekä kehitysprosessin vaiheiden ja havaintojen perusteella tehtyjen havaintojen erittelystä.

2 DSiP

DSiP ("Distributed Systems intercommunication Protocol") on Ajeco Oy:n kehittämä vikasietoinen monikanavareititysjärjestelmä. Järjestelmän tarkoitus on toimia luotettavana ja vikasietoisena tietoliikenteen reititysjärjestelmänä.

Ajecon mukaan "DSiP mahdollistaa kaikenlaisten viestintäkanavien ja -resurssien käytön rinnakkain. Monet viestintäkanavat näkyvät käyttäjälle yhtenä, kesteytymättömänä ja vankkana kanavana DSiP-järjestelmää käyttäville laitteille, ohjelmistoille ja välineille" (Ajeco 2012.)

DSiP on kehitetty vastaamaan luotettavan tiedon välityksen tarpeeseen. Nykytekniikan vaatimuksena on tiedonkulun luotettavuus sekä helppokäyttöisyys. On tärkeää, että arvokas tieto on keskeytymättömästi sekä luotettavasti saatavilla verkossa (Holmström, Rajamäki, Hult 2011, 115.)

Erityisesti tämänkaltaisista järjestelmistä hyötyvät erilaiset viranomaisorganisaatiot sekä turvallisuutta tai maanpuolustusta valvovat organisaatiot. Näillä organisaatioilla on lisäksi suuri etu käyttää useita viestintäkanavia tiedon välitykseen, sillä mikäli
tiedon välitys olisi riippuvainen vain yhdestä viestintäkanavasta, tämän kanavan ollessa pois toiminnasta tietoa ei voisi siirtää lainkaan. Holmströmin, Rajamäen sekä Hultin mukaan seuraavia arvoja voidaan käyttää viitteenä hyvälle tietojärjestelmälle:

1. Käytettävyys
2. Suorituskyky
3. Skaalautuvuus
4. Luotettavuus
5. Saatavuus
6. Laajennettavuus
7. Ylläpidettävyys
8. Hallittavuus
9. Luotettavuus ja turvallisuus

Näiden arvojen toteutuminen voidaan todeta Holmströmin, Rajamäen ja Hultin mukaan vain järjestelmän käyttöönoton jälkeen. (Holmström ym. 2011, 116.) Tämän työn tutkimuksen yhteydessä luotu DSiP-konfiguraation hallintaohjelma toimii siis mallin tasoilla seitsemän ja kahdeksan, eli ylläpidettävyyden ja hallittavuuden parissa.

2.1 Järjestelmän reitittimet

DSiP-järjestelmä rakentuu erityisten DSiP-reititinten varaan. Ne ovat laitteita, joiden tehtävänä on välittää salattua DSiP-protokollan liikennettä. Reititinten tehtävä on muodostaa tietoverkko eri osapuolten välille, ja välittää luotettavasti osapuolten haluama tieto verkkoa pitkin. Yksi reititin vastaa vain tietyn viestin lähettämisestä seuraavalle reitittimelle, ja tämän reitittimen tehtävänä on puolestaan välittää tieto taas seuraavalle, kunnes lopulta saavutetaan tiedon vastaanottaja.

2.2 Konfiguraatiotiedostojen rakenne

Reititinten konfigurointi suoritetaan erityisten konfiguraatiotiedostojen avulla. Tiedostoihin merkitään kustakin reitittimestä verkon toiminnan kannalta tarvittavat tiedot.
Reititinten konfiguraatio jaetaan eri tiedostoihin konfiguraation aihealueen mukaan. Esimerkiksi verkon rakenne pidetään konfiguraatiossa erillään reititinten yksittäisistä asetuksista.

Konfiguraatio aktivoidaan lähettämällä konfiguraatiotiedostot laitteisiin joko paikallisesti tai DSiP-verkon ylitse. Laitteiden ominaisuuksiin kuuluu konfiguraation käyttöönotto sen vastaanottamisen jälkeen. Järjestelmä on siis hallittavissa etäyhteyden kautta tällä tavalla.

3 Java

Java on Sun Microsystemsin alun perin kehittämä ohjelmointikieli. Se kehitettiin alun perin "käytettäväksi sittemmin unohdetussa kodin elektroniikkalaitteiden ohjaukseen tarkoitetussa *7-laitteessa (StarSeven). Jo alusta asti kielen tavoitteena oli olla prosessoririippumaton, jolloin samalla ohjelmointikielellä pystyttäisiin toteuttamaan sovelluksia useille eri alustoille" (Salonen 2011, luku 2).

Javaa käytetään tänä päivänä 1,1 miljardissa pöytäkoneessa, kolmessa miljardissa matkapuhelimessa sekä kaikissa Blu-ray -soittimissa. Kieltä käytetään lisäksi digitaali-TV-vastaanottimissa, tulostimissa, webcameissa, peleissä, autonavigaattoreissa, arvontapäätteissä, lääketieteellisissä laitteissa sekä pysäköintiautomaateissa. (Oracle 2011.)

3.1 Monialustaisuus

Java-kieli on suunniteltu rakenteeltaan sellaiseksi, että sitä on mahdollista kääntää eri alustoilla kuin millä sitä ajetaan. Kääntö ei tapahdu suoraan Java-koodista konekielelle, vaan Java-tavukoodiksi.

3.2 Tavukoodi

Käännön jälkeen ohjelma on tavukoodimuodossa. Tämä tarkoittaa sitä, että sitä ei voida suoraan suorittaa tietokoneen prosessorilla, vaan se on ensin muunnettava tietokoneen suorittimen eli prosessorin ymmärtämään binäärimuotoon. Binäärimuotoon muuntaminen suoritetaan Javan JVM-virtuaalikoneella. (Salonen 2011, luku 3.)

3.3 Virtualikone

Virtuaalikone on tietokoneen isäntäkäyttöjärjestelmän sisällä ajettava ohjelma, joka tarjoaa käyttöjärjestelmän kaltaiset olosuhteet toisille ohjelmille. Virtuaalikoneella voidaan suorittaa ohjelmia eri ympäristössä kuin mitä tietokoneen isäntäkäyttöjärjestelmä tarjoaa. (Caprio 2006.)

Java-virtuaalikoneen päätarkoitus on suorittaa käännettyä Java-tavukoodia sillä alustalla, jolle virtuaalikone on käännetty. Virtuaalikone myös piilottaa ohjelmoijalta suoria kutsuja käyttöjärjestelmän toimintoihin, ja tarjoaa näiden sijaan standardoidun käyttöliittymän yleisimpiin operaatioihin, kuten tiedostojärjestelmän hallintaan ja laitteiden käyttöön. Tätä arkkitehtuuria noudattamalla Java-virtuaalikoneita on kirjoitettu useille alustoille ja erilaisiin käyttöympäristöihin.

Java-Kieli on suunniteltu niin yksinkertaiseksi, että monet ohjelmoijat voivat oppia kirjoittamaan sillä sujuvasti. Java-ohjelmointikieli on sukua C- sekä C++ - kielille, mutta se on rakennettu toimimaan hieman eri tavalla. (Oracle 2011.)

3.4 Automaattinen muistinhallinta ja roskienkeruu

Merkittävä ero vanhempiin C-sukuisiin kieliin on Javan automaattinen muistinhallinta. Muistinhallinta tarkoittaa sitä tapaa, jolla ohjelmoija määrittelee tietokoneen käyttämään saatavilla olevaa muistia. Muistinhallinta voi olla automaattista tai manuaalista. Siinä missä C-sukuisissa kielissä muistinhallinta on manuaalista, eli se on jätetty ohjelmoijan vastuulle, Java käyttää automaattista muistinhallintaa, eli ohjelmoijan vastuulla ei ole määritellä erikseen käytettävän muistin hallintaa.

Javassa muistinhallinnan hoitaa erityinen roskienkerääjä. Sen tehtävänä on tarkkailla, milloin käytettäviä resursseja ei enää tarvita, ja "kerätä" ne pois, eli vapauttaa ne. Resurssien vapautuksen jälkeen niitä voidaan käyttää jonkin muun tiedon säilöntään. (Sun Microsystems Inc 2001.)

3.5 Luokat ja oliot

Java-kieli on olio-ohjelmointikieli, ja täten perustuu luokkiin ja olioihin. Luokat ovat itsenäisiä ohjelman komponentteja, jotka vastaavat reaalimaailman entiteettejä. Esimerkiksi henkilöitä käsittelevä ohjelma todennäköisesti sisältää yhtä henkilöä mallintavan luokan. Luokilla on kaksi tarkoitusta. Ensimmäinen tarkoitus on järjestellä ohjelman koodi loogisiin rakennuspalikoihin. Luokkien toinen tarkoitus on tarjota muille luokille sekä niiden sisältämää tietoa että menetelmiä muuttaa ja esittää sitä.

3.6 Muuttujat

Luokat ja oliot säilövät tietoa kentissä. Jokaisella kentällä on nimi sekä mahdollisesti jokin arvo, jonka luokka voi halutessaan tarjota toisille luokille luku-tai kirjoitusoikeuden omiin kenttiinsä. Kenttien arvo voi olla vakio tai vaihtuva. Lukuoikeudellisia kenttiä kutsutaan vakioiksi, ja luku- sekä kirjoitusoikeudellisia kenttiä muuttujiksi. Vaikka kenttien arvot voivat joskus muuttuakin, kenttien nimet pysyvät kuitenkin aina samoina. Kenttiä käytetään pääasiassa tiedon käsittelyyn liittyvinä tiedon tallennuspaikkoina.

3.7 Metodit

Tietoa käsitellään Javassa metodien avulla. Metodi vastaa suureksi osaksi matematiikan käsitettä funktiosta. Funktiolle annetaan parametriksi arvo tai joukko arvoja, ja funktio palauttaa jonkin näistä arvoista lasketun tuloksen. Javan metodit eroavat matemaattisista funktioista siten, että Javassa metodilla on aina oikeus toimia tilallisessa kontekstissa. Tämä tarkoittaa sitä, että toisin kuin matemaattisella funktiolla, metodilla on mahdollisuus lukea ja kirjoittaa muita arvoja kuin niitä, jotka se on saanut parametreikseen. Metodi voi siis esimerkiksi olla palauttamatta arvoa lainkaan, ja päivittää sivuvaikutuksena jotakin luokan muuttujista. Metodien avulla ohjelman kulkua voidaan hajottaa loogisiin kokonaisuuksiin.

Yleinen käytäntö ohjelmistokehityksessä on, että ohjelmakoodista pyritään tekemään mahdollisimman "faktoroitua". Termi tulee matematiikan tekijä-käsitteestä, jonka mukaan jotkin luvut voidaan jakaa pienempiin tekijöihin. Tällöin kaikkien tekijöiden tulo on kyseinen luku. Ohjelmistokehityksessä tekijöihin jako voidaan mieltää siten, että koko ohjelma on kaikkien sen metodien lopputulos. Tekijöihin jako hyödyttää ohjelmoijaa siten, että ohjelmakoodita tulee helppolukuisempaa ja helpommin uudelleenkäytettävää.

3.8 Kovakoodaus

Eräs metodien hyödystä on ohjelmakoodin järjestely loogisiin osioihin. Tällöin koodin sekalaiset osat voidaan sijoittaa omiin osioihinsa, ja koodista saadaan täten loogisesti järjesteltyä ja selkeää. Metodit tekevät koodista myös uudelleenkäytettävää, koska omiin metodeihinsa järjesteltyjä koodiosioita voidaan käyttää uudelleen. Mikäli metodeista on tehty yleiskäyttöisiä, niitä voidaan käyttää myös muissa konteksteissa kuin mihin ne alun perin ovat sopineet.

3.9 Javan tyyppijärjestelmä

Jokaisella kentällä ja metodilla on Javassa jokin tyyppi. Tyyppi voi esimerkiksi olla merkkijono tai kokonaisluku, kuten "Martti" tai 5. Kentän tyyppi rajoittaa kenttään sijoitettavia arvoja siten, että niiden täytyy kaikkien olla samantyyppisiä kuin kenttä itse. Metodin tyyppi rajoittaa metodille parametreiksi annettavien muuttujien tyypin. Esimerkiksi murtolukuja käsittelevä metodi hyötyy siitä, että sen ilmoitetaan vastaanottavan parametreiksi vain murtolukuja. Java-ympäristö pitää kirjaa tyypeistä, ja huolehtii siitä, että tyyppijärjestelmää noudatetaan. Javan tyyppijärjestelmää kutsutaan tämän ominaisuutensa vuoksi vahvaksi tyyppijärjestelmäksi.

Tyyppejä voidaan myös erityistilanteissa muuntaa toisiksi tyypeiksi. Tyyppien muuntojärjestelmiä on kahdenlaisia, staattisia sekä dynaamisia. Staattinen tyypitys on hieman hitaampaa kirjoittaa ja suunnitella kuin sen vastakohta, dynaaminen tyypitys. Staattisen tyypityksen hyötynä on kuitenkin se, että se auttaa ohjelmoijaa korjaamaan ohjelmassa esiintyvät virheet sen kääntämisen aikana. Dynaaminen tyypitys pakottaa virheiden etsimisen tapahtuvan ohjelman suorittamisen aikana, mutta se puolestaan on nopeampaa kirjoittaa. (Oracle 2011.)

3.10 Aksessorit

Hyvin yleinen käyttö kentille ja metodeille onkin niin sanottujen aksessorien käyttö. Aksessori on metodi, jonka tehtävänä on toimia käyttöliittymänä luokan kenttien asettamiselle tai hakemiselle. Luokat haluavat lähes aina varmistaa, että niissä oleviin kenttiin sijoitetaan oikeaa, validia, tietoa. Tällöin luokka päättää olla sallimatta suoraa luku- ja kirjoitusoikeutta kenttäänsä muilta luokilta, ja käyttää vaihtoehtona kahta metodia.
Esimerkiksi kentälle "osoite" voidaan luoda aksessorit asetaOsoite() sekä haeOsoite(). Luokka voi tällöin sijoittaa aksessoreihin haluamaansa tarkistuskoodia, ja täten varmistua siitä, että kentän arvoa käsitellään oikein.

3.11 Imperatiivinen ohjelmointi

Javan tukemaa ohjelmointitapaa kutsutaan imperatiiviseksi ohjelmointitavaksi tai viitekehykseksi. Imperatiiviseen tapaan kuuluvat tilallisten muutosten sekä ohjelman suoritusjärjestyksen korostaminen. Ohjelmoija keskittyy imperatiivisessa viitekehyksessä algoritmien (tehtävien) suorittamiseen sekä ohjelman tilan muutosten seuraamiseen. Ohjelman suoritus, eli vuonohjaus, toteutetaan toistuvilla silmukoilla, ehtorakenteilla sekä metodikutsuilla. (Microsoft 2012.)

Tilalliset muutokset tarkoittavat sitä, että ohjelman eri suoritusvaiheissa samoissa muuttujakentissä voi olla eri arvoja. Suoritusjärjestys on tavallisesti peräkkäistävä, eli koodissa aiemmin esiintyvät lauseet suoritetaan ennen myöhemmin esiintyviä lauseita.

Vuonohjaus tarkoittaa sitä, miten ohjelman suoritusta voi ohjata muutoin kuin peräkkäisillä käskyillä. Tästä esimerkkinä ovat silmukat, joissa olevaa koodia voidaan toistaa useita kertoja; ehdot, jolloin osa koodia voidaan suorittaa vain jos jokin ehto täyttyy; sekä metodikutsut, eli metodien suorittaminen, jolloin jokin luokan metodi suoritetaan sille annetuilla parametreilla.

3.12 Sivuvaikutukset

Ohjelmointikielissä sivuvaikutus-käsitteestä puhuttaessa tarkoitetaan sitä, että ajettava ohjelma suorittaa jonkin sellaisen operaation, joka liittyy reaalimaailman tilaan. Tällaisia operaatioita ovat esimerkiksi jonkin tiedon hakeminen tietokoneen kiintolevyltä, äänikortin tai näytönohjaimen käyttö äänen tai kuvan esittämiseksi sekä verkko-operaatiot. Ohjelmointikielen toiminta on muutoin täysin loogisesti ennakoitavissa, mutta sivuvaikutuksia aiheuttavan tai niistä riippuvaisen ohjelman suorituksen tilaa ei voida ennustaa. Tällöin voidaan sanoa, että ohjelman kulkuun on vaikuttanut jokin sen koodin ulkopuolinen asia, eli sivuvaikutus.

Sivuvaikutuksien vastakohta ovat niin sanottu puhdas koodi. Koodin puhtaus tarkoittaa sitä, että sen toiminta ei riipu mistään ulkopuolisesta seikasta. Tästä johtuen voidaan esimerkiksi sanoa, että puhdas funktio palauttaa aina saman tuloksen riippumatta sen parametreista. Esimerkkejä puhtaista funktioista ovat luvun neliöjuuren laskeva funktio tai tässä tutkimustyössä käytetty reitittimen jonkin ominaisuuden arvon oikeellisuuden tarkistava funktio.

Mikäli mitään sivuvaikutuksia ei sallittaisi, tietokoneohjelma ei pystyisi esimerkiksi esittämään käyttäjälle minkäänlaista tulosta,koska tuloksen näyttäminen näyttölaitteen kautta muuttaisi reaalimaailman tilaa. Ohjelma ei myöskään pystyisi toimimaan minkään sille ulkoisesti syötettyjen parametrien perusteella, vaan tekisi aina saman ennalta määritellyn asian.

Sivuvaikutukset ovat siis välttämättömiä järkevien tietokoneohjelmien toiminnan kannalta. Niiden epäjärjestelmällinen tai liiallinen käyttö tuottavat kuitenkin ohjelmoijalle ylimääräistä päänvaivaa.

Sivuvaikutusten käytön mukana helposti tuleva haitta on se, että koska sivuvaikutuksia ei imperatiivisissa ohjelmointikielissä, kuten Javassa, tavallisesti rajata toimimaan puhtaasta koodista eristettynä, ohjelman tilan muutokset ovat vaikeammin havaittavia. Koska ohjelmointikieli jättää tämän seikan ohjelmoijan vastuulle, tämä johtaa helposti ohjelman resurssien hallinnan epäjärjestelmällisyyteen. Tämä puolestaan voi johtaa siihen, tietoa käsitellään odottamattomissa tilanteissa väärin.

3.13 Poikkeukset

Javassa, kuten monissa muissakin oliokielissä, on mahdollista tehdä varotoimia sille, että jokin osa ohjelman koodin suorituksesta epäonnistuu. Tällainen tilanne voi tapahtua sekä puhtaassa, aidossa koodissa, että sivuvaikutuksia käyttävässä, epäpuhtaassa koodissa.

Puhdas, eli sivuvaikutuksista vapaa, koodi voi epäonnistua esimerkiksi siten, että jakolaskua suorittava funktio käyttää jakajana lukua nolla. Epäpuhdas koodi voi puolestaan epäonnistua tilanteessa, jossa ohjelman suoritus riippuu jostakin Internetin ylitse haettavasta tiedosta, kuten esimerkiksi junan lähtöajasta.
Mikäli tässä tilanteessa yhteyttä tiedon saavalle palvelimelle ei voida muodostaa, ohjelman suoritusta ei voida jatkaa, tai sitä jatketaan olemattomalla tiedolla, joka taas voi aiheuttaa myöhemmän osan ohjelmakoodin epäonnistumisen.

Poikkeuksia on kahdenlaisia, käsiteltyjä ja käsittelemättömiä. Käsitellyt poikkeukset ovat sellaisia, joihin ohjelmoija on opettanut ohjelman varautumaan etukäteen. Tällöin ohjelma voi, virheen vakavuudesta riippuen, jatkaa toimintaansa, eikä sitä tarvitse sulkea kokonaan.

Käsittelemättömät poikkeukset ovat sellaisia, joihin ohjelmoija ei ole varautunut. Käsittelemätön poikkeus esiintyy esimerkiksi tilanteessa, jossa positiivista kokonaislukua odottavalle neliöjuurta laskevalle metodille annetaan parametriksi negatiivinen kokonaisluku, ja metodin laskutoimitus epäonnistuu, koska negatiivisilla luvuilla ei ole neliöjuuria reaalilukujen joukossa. Koska ohjelmoija ei ole varautunut tähän erikoistilanteeseen, ohjelma kohtaa käsittelemättömän poikkeuksen.

Tutkimuksessa esiintyvä käsittelemätön poikkeus voisi aiheutua tilanteessa, jossa yritetään lukea DSiP-laitteen konfiguraatiotiedosto. Ohjelma olettaa tiedoston sijaitsevan tietyssä sijain-

nissa, mutta tiedostoa ei löydykään kyseisestä sijainnista. Tällöin ohjelma ilmoittaa virhetilanteesta heittämällä poikkeuksen. Mikäli ohjelmoija ei ole olettanut näin käyvän, hän ei ole myöskään kirjoittanut poikkeusta käsittelevää koodia, ja tässä tapauksessa ohjelman suoritus keskeytyy kyseiseen virheeseen.

Java-kielessä ohjelmoijan on pakko merkitä metodien määrittelyn yhteyteen, mikäli kyseinen metodi heittää poikkeuksen. Tämä rajoitus on luotu ohjaamaan ohjelmoijia hyvän käytännön noudattamiseen. Merkintätapa selkeyttää koodin lukemista, sillä mikäli heitettävien poikkeusten tyyppejä ei määriteltäisi, niitä voisi olla mahdotonta havaita, ellei jo valmiiksi tuntisi koodia.

3.14 Käyttöliittymä

Tietokoneohjelmia suunnitellaan jatkuvasti eri tarkoituksiin. Monien ohjelmien tarkoitus on tarjota käyttäjälle toimiva työkalu jonkin asian suorittamiseksi. Osa ohjelmista, kuten pelit, ovat viihteellisiä. Osa ohjelmista, kuten haittaohjelmat, on jopa suunniteltu toimimaan käyttäjän tahtoa vastaan.

Tämän työn tutkimukseen liittyy toimivan, työkalun roolissa olevan ohjelman kehitys. Työkaluohjelmalle on tavallista tarjota käyttäjälle interaktiivinen käyttöliittymä, jonka kautta käyttäjä voi vaikuttaa ohjelman toimintaan. Yleinen ja tunnistettava käyttöliittymätyyppi on graafinen käyttöliittymä.

3.15 Graafinen käyttöliittymä

Graafinen käyttöliittymä piirretään tietokoneeseen liitetylle näyttölaitteelle kuvapiste kerrallaan. Graafisen käyttöliittymän tarkoitus on tarjota käyttäjälle visuaalinen, intuitiivinen näkymä ohjelman tilaan ja toimintoihin. Käyttöliittymä koostuu elementeistä, joihin kuuluvat valikot, painikkeet, listat, tekstikentät sekä aputekstit. Käyttäjä voi välittää tietoa ohjelmalle elementtien kautta. Tiedon välitykseen voi esimerkiksi kuulua ohjelmalle haluamiensa tietojen syöttäminen, jonkin ohjelman ominaisuuden aktivoiminen tai ohjelman tilan tarkistus. Käyttöliittymissä olevia elementtejä käytetään toistuvasti useissa paikoissa. Tämän vuoksi on tapana tallentaa niiden toteutus erityisiin grafiikkakirjastoihin.

3.16 Ohjelmointikirjasto

Ohjelmointikirjasto on kokoelma koodia, joka on kirjoitettu uudelleenkäytettävään muotoon. Kirjasto on tarkoitettu ohjelmoijan hyödyksi. Ohjelmoija voi kirjastoa käyttämällä lisätä ohjelmaansa kirjaston tarjoamia toimintoja, eikä ohjelmoijan tällöin tarvitse kirjoittaa toiminnallisuuksia uudelleen itse. Esimerkiksi grafiikkakirjastoja käyttämällä ohjelmoija voi lisätä ohjelmaansa graafisen käyttöliittymän valmiiksi määriteltyjä elementtejä.

4 Versionhallinta

Versionhallinta on työkalu ohjelmakoodin hallintaan. Se pyrkii vastaamaan ohjelmakoodin hallinnassa usein esiintyviin tarpeisiin, kuten aikaisempiin koodin versioihin palaamiseen, koodin jaettavuuteen sekä useiden ominaisuuksien yhtäaikaisen kehityksen helpottamiseen.

4.1 Versionhallinnan käyttöperusteet

Versionhallinnan käyttö perustuu siihen, että koodiin tehdyt toimivat muutokset tallennetaan versionhallinnan piiriin omina kokonaisuuksinaan. Jokaiseen versionhallintaan tallennettuun kokonaisuuteen liitetään kuvaus muutoksesta, sekä erä yleistä metatietoa, kuten aika sekä muutoksen tehneen käyttäjän tietoja. Versionhallinta pitää kirjaa muutoksista, ja tarjoaa sen perusteella em. toimintoja sen käyttäjille.

Versionhallintaa käytetään ohjelmoinnissa ohjelmien lähdekoodin kehityksen hallintaan. Se tarjoaa työkalut ohjelmakoodin hallintaan ja kehityksen seurantaan. Sen avulla voidaan tallentaa koodista eri versioita, palata vanhoihin versioihin, sekä jakaa ja julkaista koodia toisille kehittäjille.

4.2 Versiohistorian tiloihin palaaminen

Ohjelmistokehityksessä aikaisempiin versioihin palaaminen on tärkeä ominaisuus. Sitä voidaan käyttää varotoimena sille, että mikäli jokin osa ohjelmistosta ei jostain syystä enää toimi, saadaan versionhallinnan kautta palautettua ohjelmasta aiemmin toiminut versio. Versionhallinnan kautta voidaan myös saada yleiskäsitys ohjelmiston kehityksestä. Siitä nähdään, missä vaiheessa kukin muutos on koodiin tehty, ja tämän ja muiden tietojen perusteella voidaan hallita ohjelmiston kehitystä.

4.3 Koodin jaettavuus

Koodin jaettavuus tuo etuna sen, että koodin siirtely usean työpisteen tai median välillä helpottuu tai mahdollistuu. Versionhallinta tarjoaa tavallisesti myös tuen erilaisten siirtomekanismien käyttämiseen. Se mahdollistaa eri siirto- ja säilytysmedioiden käyttämisen siten, että osa koodivarastoista voi sijaita eri käytettävän tiedostojärjestelmän kansioissa ja osa varastoista eri Internetin palvelimilla.

4.4 Yhtäaikainen kehitys

Versionhallinta on myös työkalu yhtäaikaisen kehitykseen. Versionhallinta antaa käyttäjilleen mahdollisuuden säilyttää vakaa versio työstään erillään yhden tai useamman kehitysversion kanssa. Tavallinen käyttötapa versionhallinnalla onkin sellainen, jossa yksi kehityshaara sisältää vain vakaita ja toimivia julkaisuja, ja toimiville julkaisuille pohjautuvia ominaisuuksia kehitetään omilla kehityshaaroillaan. Tällöin kehittäjien muutokset, jotka välillä saattavat haitata ohjelman toimivuutta, eivät vaikuta toisten kehittäjien koodin toimivuuteen.

4.5 Git-versionhallinta

Git on lähdekoodin lisensoinniltaan vapaa, hajautettu versionhallintajärjestelmä. Versionhallinta tarjoaa yleensä myös mahdollisuuden kehittää ohjelmakoodista erilaisia versioita samanaikaisesti. Kehittäjä voi siirtyä näiden versioiden välillä, ja versionhallinta vastaa siitä, että kaikki kooditiedostot pysyvät oikeassa tilassa. Git tukee haaramallikehitystä hyvin. Sen tarjoamat kehityshaarat ovat kevyitä, ja niitä on helppoa ja nopeaa käsitellä.

4.6 Gitin haaramalli ohjelmointityökaluna

Yleinen Git-kehitystapa onkin sellainen, jossa yksi tai useampi kehityshaara varataan yksinomaan toimiville, julkaisukelpoisille versioille, ja loput haarat keskittyvät yksittäisten ominaisuuksien lisäämiseen tai jälkikäteen löytyneiden virheiden korjaamiseen. Kun muiden haarojen tavoite on saavutettu, ne voidaan yhdistää pää-tai "master"-haaraan.

Haaramallia käytettäessä voidaan versionhallinnan piirissä olevia tiedostoja palauttaa aikaisempiin versioihin. Tällöin kehittäjän ei tarvitse huolehtia siitä, että jokin osa toimivasta tai kokeellisesta koodista katoaisi vaikka se poistettaisiinkin tiedostoista.

Ohjelmakoodin siirrossa on aina vähintään kaksi osapuolta, lähettäjä sekä vastaanottaja. Tapahtumassa koodi siirretään lähettäjän tietokoneelta vastaanottajalle. Sekä lähettäjä että vastaanottaja voivat olla joko ihmisiä tai tietokoneita. Kun vastaanottaja on vastaanottanut

ohjelmakoodin, sanotaan koodivarastojen olevan synkronoidut, eli samanaikaistetut. Niissä molemmissa on siis täsmälleen sama ohjelmakoodi samassa historiallisessa järjestyksessä.

Gitin tapauksessa vastaanottaja ja lähettäjä voivat synkronoida yhden tai halutessaan useamman haaran. Tämä mahdollistaa sen, että eri kehittäjät voivat tarkasti hallita sitä, että he julkaisevat muille kehitystiimissä oleville kehittäjille vain toimivaa koodia, sekä sen, että he voivat halutessaan säilyttää kokeelliset lisäyksensä koodiin vain omalla työpisteellään.

Gitin koodin jakamisen toimintamalli on joustava asiakas-palvelin -malli. Git-ympäristössä on aina olemassa varastoja, jotka sisältävät versioituja tiedostoja. Gitin mallissa ei kuitenkaan välttämättä tarvita yksinomaan Gitille varattua palvelinta koodin säilöntään, vaan mikä tahansa tietokone voi toimia sekä asiakkaana että palvelimena koodia siirrettäessä.
Gitin käyttö ei myöskään vaadi jatkuvaa Internet-yhteyttä koodivarastojen synkronointiin, kuten useat versionhallintajärjestelmät, vaan varastot voidaan synkronoida silloin kuin se parhaiten kehittäjille sopii.

5 GNU/Linux – käyttöjärjestelmä

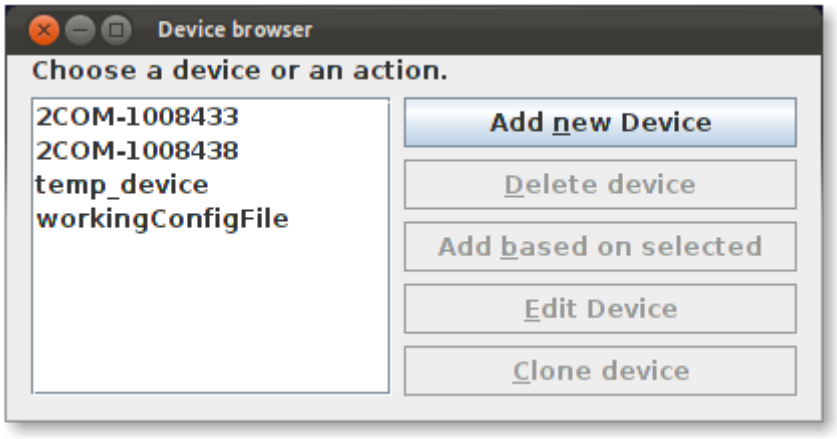
Tutkimuksen pääkehitysalustaksi valittiin ilmainen ja vapaa GNU/Linux - käyttöjärjestelmäpohja, sillä se tarjosi monia hyviä työkaluja ohjelman kehittämistä varten. Käytettyjä työkaluja olivat työasemien käyttöjärjestelmä, palvelinten käyttöjärjestelmä, tekstieditori sekä em. versionhallintatyökalut työasemilla ja palvelimilla. Lisäksi GNU/Linux -käyttöjärjestelmää käytettiin yhtenä ohjelmantestialustana.

6 DSiP-ohjelman rakenne

Ohjelman käyttöliittymässä on käyttäjälle kaksi näkyvää osaa. Osat ovat laiteselain ja laitemuokkain eli editori. Näissä näkymissä käyttäjä voi hallita laitteita useiden laitteiden tasolla sekä lähemmin yhtä laitetta kerrallaan.

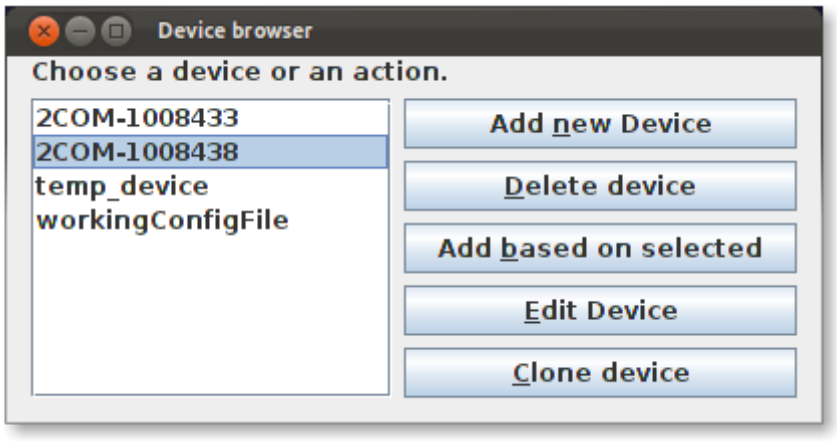
Laiteselaimen tarkoitus on nostaa abstraktiotasoa siten, että käyttäjä voi keskittyä yksittäisten laitteiden ylläpitoon. Selain käsittelee em. tiedostoja käyttäjän kannalta täysin näkymättömästi. Ohjelma myös luonnollisesti pitää huolen siitä, että tiedostojen keskinäiset ja paikoin monimutkaiset suhteet säilyvät oikeanlaisina. Tiedostojen välillä on monia riippuvuussuhteita, ja yhdellä muutoksella voi olla sivuvaikutuksia muiden tiedoston oikeellisuuteen.

Laiteselain konfiguroidaan syöttämällä mainConfig.cfg - nimiseen tiedostoon laitetiedostojen hakemisto sekä sarjanumero- että verkon solmutiedostojen sijainnit.



Kuva 1: Laiteselaimen perusnäkymä

Laiteselain listaa kaikki laitetiedostojen hakemistosta löytyneet laitetiedostot sarjanumeron perusteella. Käyttäjä voi valita ohjelman tarjoamista vaihtoehdoista tehtävän toimenpiteen. Käyttäjälle tarjotaan seuraavat vaihtoehdot:



Kuva 2: Laitteen valittuaan käyttäjä saa lisää toimintoja käyttöönsä

6.1 Laitteen lisäys ja poisto

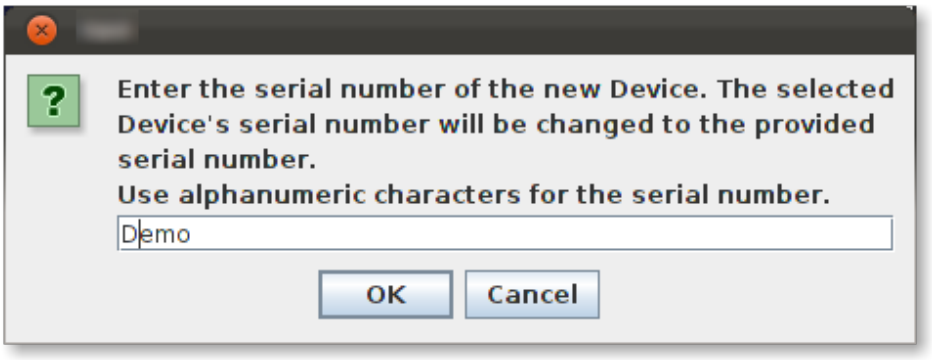
Laitteen lisäyspainikkeesta uusi laite lisätään muiden joukkoon. Käyttäjälle avataan laitemuokkain, johon hän voi syöttää kaikki uuden laitteen tiedot. Kaikki käyttäjän antamat arvot tarkistetaan ja kirjoitetaan oikeisiin tiedostoihin. Arvoista tarkistetaan niiden oikeellisuus, sekä se, että annetut arvot eivät ole ristiriidassa muiden konfiguraation laitteiden kanssa.

Laitteen poistopainikkeesta listasta valittu laite poistetaan konfiguraatiosta. Kaikki konfiguraatiotiedostot, jotka viittasivat tähän laitteeseen jollain lailla, kirjoitetaan uudelleen siten, että niissä ei ole viittausta kyseiseen laitteeseen.

6.2 Uuden laitteen lisäys käyttäen vanhaa laitetta pohjana uudelle

Listasta valittu laite otetaan pohjaksi uudelle laitteelle. Käyttäjää pyydetään syöttämään uuden laitteen tiedot kuten laitetta lisätessäkin, mutta nyt erona on, että laitemuokkaimen kentät ovat esitäytetyt listasta valitun laitteen tiedoilla.

6.3 Laitteen sarjanumeron vaihto



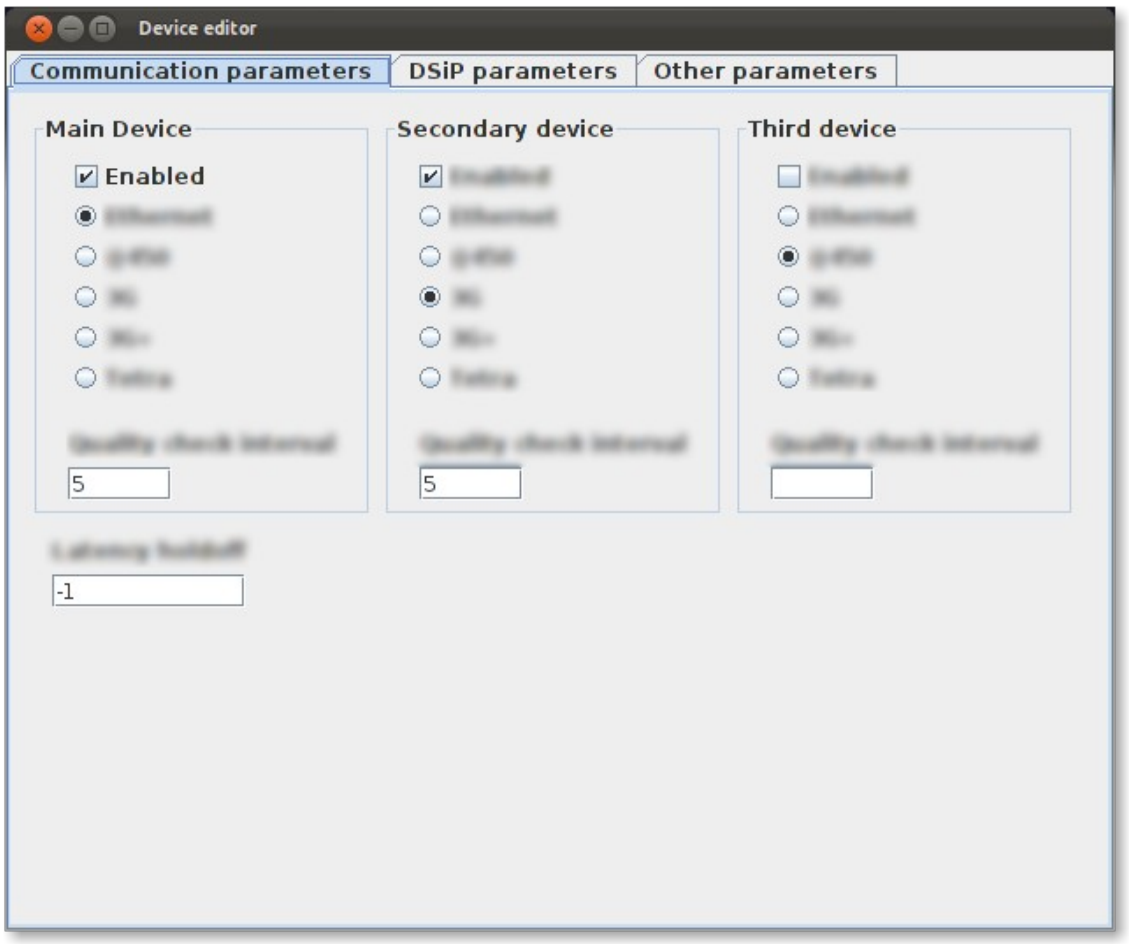
Kuva 3: Käyttäjältä kysytään uuden laitteen sarjanumero

Valitun laitteen sarjanumero vaihdetaan käyttäjän antamaan sarjanumeroon. Kaikki laitteeseen viittaavat konfiguraatiotiedostot kirjoitetaan uudelleen vastaamaan uutta sarjanumeroa.

6.4 Laitteen muokkaus

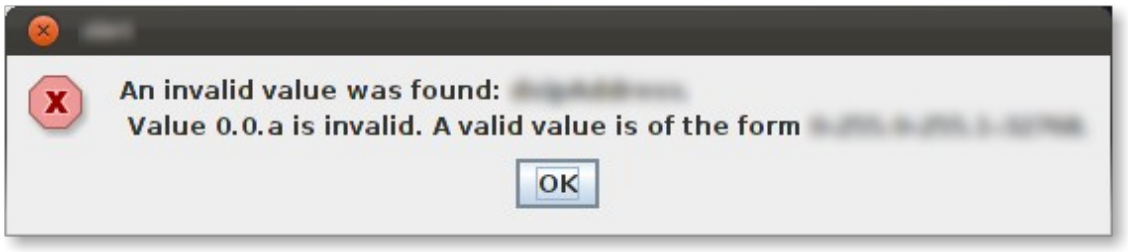
Käyttäjälle avataan laitemuokkain, jossa on esitäytettyinä listasta valitun laitteen tiedot. Käyttäjä voi muokata laitteen tietoja, ja tallentaa tai hylätä muutokset. Mikäli käyttäjä tallentaa laitteen konfiguraation, suoritetaan tarpeelliset tarkistukset annetuille arvoille ja kirjoitetaan kaikki laitteeseen viittaavat tiedostot uudelleen vastaamaan uusia tietoja. Aloitusnäkymässä mitään toimintoja ei voida suorittaa. Käyttäjä saa laitetiedoston valitsemalla näkyviin sille tehtävät toiminnot. Valitsemalla jonkin toiminnon laiteselain piilotetaan käyttäjältä, ja tilalle avataan laitemuokkainnäkymä.

6.5 Laitemuokkain



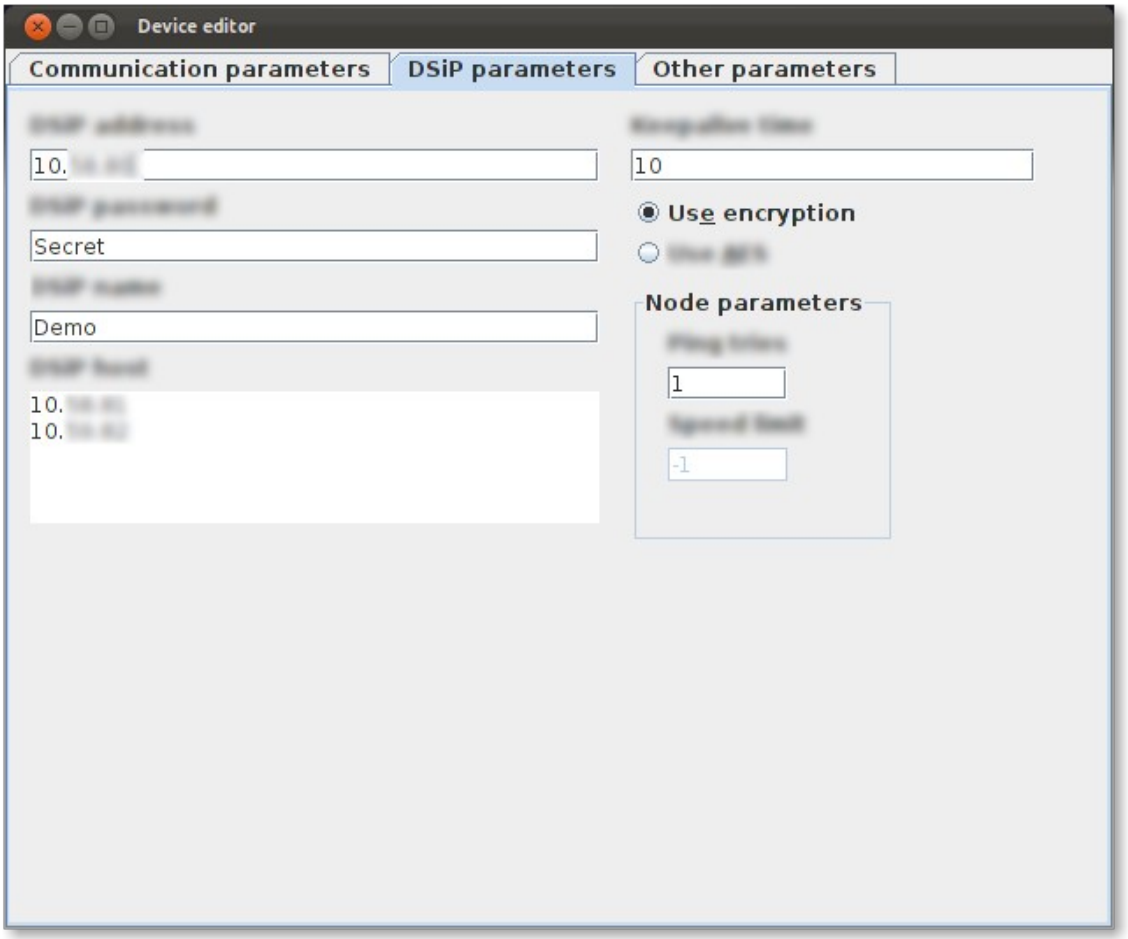
Kuva 4: Laitemuokkaimen kommunikaatioparametrinäkymä

Ohjelman toinen näkyvä osa on laitemuokkain eli laite-editori. Editori näytetään käyttäjälle kun käyttäjä on valinnut jonkin laiteselaimen tarjoaman toiminnon. Editori mallintaa kaikki yhden laitteen konfiguraation asetukset. Se myös tarkistaa, ettei mihinkään sen kenttään syötetä epäkelpoa tietoa. Esimerkki epäkelvon tiedon syöttämisestä on numeeriseen kenttään syötetyt kirjainmerkit. Tällaisessa tilanteessa ohjelma tunnistaa ja raportoi virheen käyttäjälle. Käyttäjä saa mahdollisuuden korjata virheen ennen kuin laitteen konfiguraatio voidaan tallentaa.



Kuva 5: Mikäli käyttäjä syöttää kelpaamatonta tietoa, ohjelma pyytää käyttäjää syöttämään tiedon uudelleen

Editori koostuu kolmesta asetusryhmiin jaetuista näkymistä. Näkymät sisältävät kaikki yhden laitteen konfiguraatioon kuuluvat asetukset. Asetukset ovat jaettu kolmeen ikkunan välilehteen, nimiltään yhteysparametrit, DSiP-parametrit sekä muut parametrit.



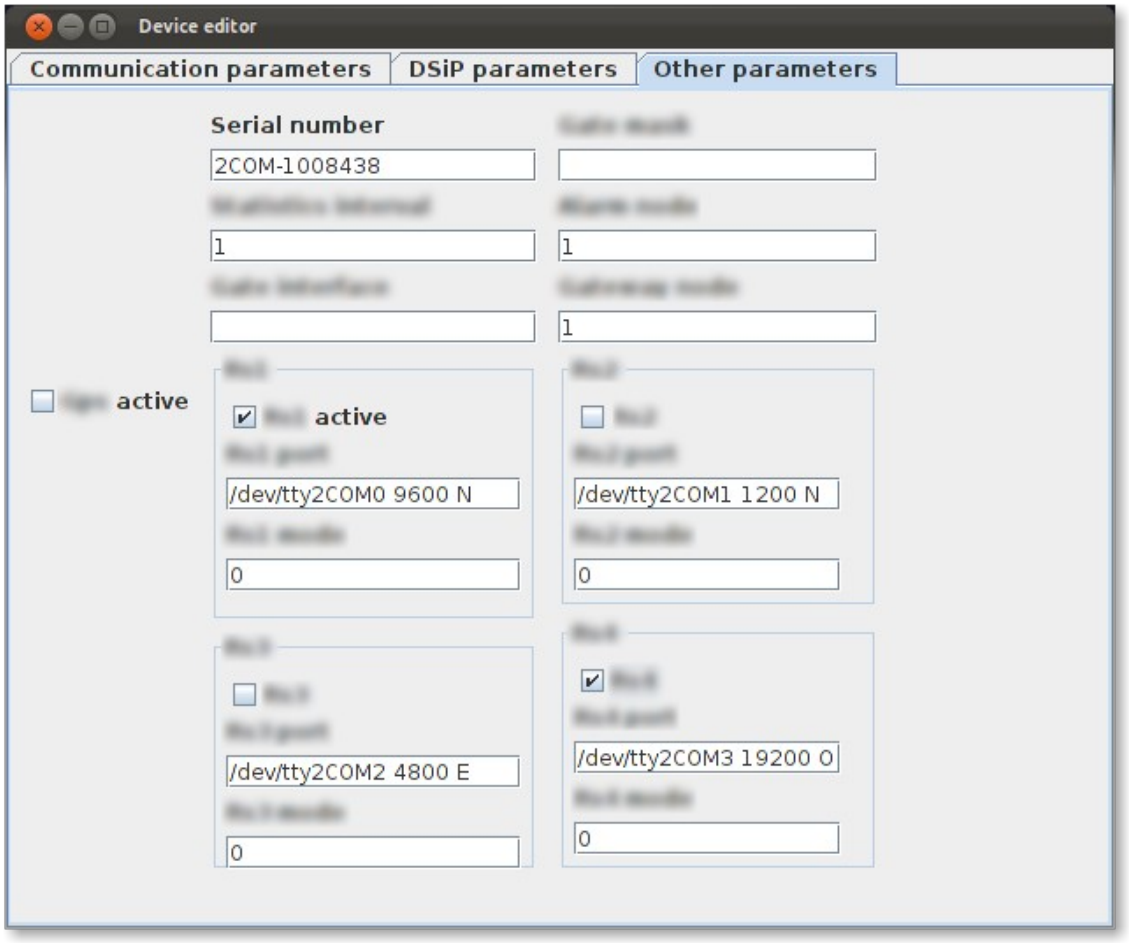
Kuva 6: Laitemuokkaimen DSiP-parametrit -näkymä

Kenttien tyypit on valittu niitä vastaavien arvojen perusteella. Tosi / taru -tyyppiset arvot on toteutettu valintaruuduilla, rajatut arvot, esimerkiksi rajauksella yhdestä viiteen, on toteutettu radionapeilla. Loput kentät ovat vapaita tekstikenttiä.

Ohjelmassa on otettu huomioon ohjelman käyttöliittymälle oleellinen ominaisuus, eli käyttäjän syöttämien arvojen tarkistus. Tämä on tärkeää kahden ohjelman käyttötarkoitukseen liittyvän komponentin toiminnan kannalta. Oikeanlaisia arvoja tarvitsevat sekä ohjelma itse että DSiP-reitittimet ja -laitteet.

Aiempien kenttien tyyppien arvojen tarkistus hoituu yleensä helposti, sillä kenttiin ei voi syöttää väärää tietoa. Tarkistettavaa on ainoastaan se, että eri kentät eivät ole ristiriidassa keskenään. Vapaat tekstikentät täytyy tarkistaa sekä syötteen että asiayhteyden kannalta.

Syötteen tarkistus on tarpeellista siinä tapauksessa, että käyttäjä antaa ohjelmalle liian pitkän syötteen. Asiayhteyden tarkistus on puolestaan tarpeellista, sillä käyttäjä voi sekä syöttää kenttään vääränlaista tietoa, että tietoa, joka on ristiriidassa jonkin muun kentän tai konfiguraation laitteen kanssa.



Kuva 7: Laitemuokkaimen "muut parametrit" -näkymä

Kolmas ohjelman vastuulla oleva tarkistus on tarkistaa, että käyttäjän syöttämä tieto ei ole ristiriidassa muiden DSiP-laitteiden konfiguraatioiden kanssa. Esimerkiksi määritellessä DSiP-verkon osoitetta täytyy tarkistaa, ettei millään muulla tunnetulla laitteella ole jo kyseistä osoitetta.

7 Ohjelman tekninen rakenne

Tässä osassa käsittelen Java-kielen tarjoamia ominaisuuksia ja niiden hyödyntämistä ohjelman kehityksessä. Ohjelma rakentuu lukuisista Java-luokista. Osa luokista on toteutettu olio-ohjelmoinnin perinnällä, ja osa luokista on itsenäisiä luokkia.

7.1 ConfigFile – luokka

Tiedostoja käsittelevät luokat periytyvät ConfigFile - luokasta. Tämä sisältää metodeja tiedoston lukemiseen sekä kirjoittamiseen. Se antaa myös mahdollisuuden hakea Ajecon konfiguraatioformaattiin merkittyjä arvo-avain - suureita.

Omille konfiguraatioon kuuluville tiedostotyypeille on ConfigFile -luokasta periytyvät Device, NodeFile sekä SerialNumberFile – luokat. Nämä tarjoavat ulkomaailmalle mahdollisuuden avata jokin tiedosto, tutkia sen sisältö sekä hallita DSiP-reititinympäristön konfiguraatiota yksi laite kerrallaan.

Luokat on toteutettu sillä tavalla, että luokan käyttäjälle tarjotaan pääsy kaikkiin tiedostoille ominaisiin muuttujiin. Muuttujia käsitellessä suoritetaan tiukkoja tarkistuksia siitä, että ympäristön konfiguraatio säilyy ehjänä mikäli muutos sallitaan. Jos muutos ei aiheuta tuhoisia sivuvaikutuksia, kuten uniikeiksi tarkoitettujen osoitearvojen päällekkäisyyksiä, ympäristössä, sallitaan muuttujan muutos.

Virheidentarkistus hyödyntää Java-kielen poikkeuksienkäsittelyä. Useimmista virheistä heitetään poikkeus, jonka luokan käyttäjä voi kopata. Poikkeus sisältää tietoa muuttujasta, jolle arvoa ei voitu asettaa, sekä ohjeita sopivan arvon asettamiseen.

ConfigFile-luokasta periytyvät luokat saavat mahdollisuuden tarkistaa, että konfiguraatiotiedosto on oikein määritelty. Tämä suoritetaan antamalla luokalle lista sallituista avaimista, joita tiedostossa saa esiintyä. Luokalle on mahdollista antaa nämä avaimet myös ulkoisesta tiedostosta. Mikäli tiedostossa esiintyy muita kuin sallittuja avaimia, käyttäjälle näytetään tästä varoitus.

7.2 Solmu- ja sarjanumeroluokat

NodeFile - luokka sisältää DSiP-verkon solmujen tiedot. Sen kautta luokan käyttäjä voi lisätä ja poistaa solmuja, sekä päästä niiden toimintaan myös suoraan käsiksi hienosäätöä varten.

Node -eli solmuluokka mallintaa yhtä solmua verkossa. Solmuluokkia kuuluu useita NodeFileen. Jokaisen solmun tietoja voi säätää, ja arvot tarkistetaan samaan tapaan kuin Device-luokkaa käytettävissä laitetiedostoissakin.

SerialNumberFile-luokka mallintaa kaikkien DSiP-ympäristön laitteiden sarjanumeroita. Sen vastuulla on kerätä sarjanumerot kaikista niistä laitteista, joista ympäristön konfiguraatiotiedostot löytyvät, sekä tarjota luokan käyttäjille sarjanumeroiden lisäys-, poisto-, luku- sekä tallennusmahdollisuudet.

7.3 Device- eli laiteluokka

Ohjelman tärkein taustaluokka on laitetiedostoa vastaava luokka Device. Luokan tarkoitus on lukea kaikki 36 yhteen laitteeseen liittyvää konfiguraatiotiedostoon kirjoitettua muuttujaa, sekä varmistaa, että nämä saavat lailliset arvot. Osalla muuttujista on ulkoiset määritykset siitä, mitä arvoja niillä voi olla. Tällaisissa tapauksissa luokan toiminnan täytyy olla erityisen tarkkaa, jotta DSiP-ympäristön kaikkien laitteiden konfiguraatio pysyy ehjänä.

8 Java-kielen tarjoamia ominaisuuksia

Käsittelen tässä osassa kehityksen aikana havaittuja Java-kielen tarjoamia ominaisuuksia ohjelmoijan näkökulmasta. Kieli ja kehitys- sekä suoritusympäristö tarjoavat ohjelmistokehityksen kannalta monia hyödyllisiä ominaisuuksia. Tämän lisäksi on otettava huomioon kielen tarjoamat rajoitukset, ja arvioitava näiden merkitys koko ohjelmistokehitysprojektille.

8.1 Monialustaisuus

Javan tarjoama suuri hyöty oli monialustaisuus, eli mahdollisuus ajaa samaa koodia eri käyttöjärjestelmillä ilman mitään muutoksia. Tästä oli hyötyä testatessa koodia eri alustoilla. Testaukseen käytettiin GNU/Linux-, Windows- sekä OS X -alustoja. Koodi toimi kaikilla samalla tavalla ilman mitään muutoksia. Käyttöjärjestelmien välillä havaittiin ainoastaan pieniä eroja ohjelman graafisessa käyttöliittymässä. Eri käyttöjärjestelmien Java-toteutukset piirtävät ikkunat ja näiden ohjaimet hieman eri tavalla. Tämän pinnallinen ero oli kuitenkin ainut havaittu erilaisuus käyttöjärjestelmien välillä.

8.2 Kattavat kirjastot

Java-ympäristöön tärkeänä osana kuuluvat sen ohjelmointikirjastot. Kirjastoja on useita, ja ne on suunnattu eri käyttötarkoituksiin.
Esimerkiksi tähän tutkimukseen liittyvän DSiP-konfiguraatioeditorin kehitykseen valittiin yleiskäyttöinen Java SE (Standard Edition). Muita kirjastojen käyttökohteita ovat yritys- ja Web-käyttöön suunnattu Java EE (Enterprise Edition) sekä sulautetuille laitteille suunnattu Java ME (Micro Edition).

Kirjastot tarjoavat kattavan kokoelman ratkaisuja yleisiin ohjelmoinnin ongelmiin. Tämän opinnäytetyön kannalta hyödyllisimpiä kirjastojen tarjoamia ratkaisuja liittyivät graafisen käyttöliittymän rakentamiseen sekä tietynlaisten IP (Internet Protocol) -osoitteiden ja aliverkon peitteiden tarkistukseen.

8.3 Aksessorien käyttö

Javan aksessorit eivät tarjonneet suurta hyötyä suhteessa niihin vaadittavaan työmäärään. Niitä käytettiin syötettäessä laitteisiin uusia arvoja. Aksessorien vastuulla oli laitteiden arvojen tarkistus eri edellä mainituissa asiayhteyksissä. Niiden kirjoittaminen muuttui kuitenkin nopeasti hyvin työlääksi, sillä yhdellä laitteella oli peräti 36 määriteltyä kenttää. Kun nämä kaikki toteutetaan ensin kenttänä, sen jälkeen haku- ja asetus-aksessoreina, tulee kirjoitettavaa 36*3==108 osan verran.

Tässäkään paljoudessa ei itsessään ole muuta vikaa kuin että osa siitä on tehty ohjelmoijalle työlääksi kirjottaa. Javassa aksessorit tulee toteuttaa erillisinä metodeina, esimerkiksi GetDeviceName() ja SetDeviceName(). Näitä luokan käyttäjä voi käyttää syntaksilla device.GetDeviceName() tai device.SetDeviceName("uusi nimi").

Esimerkiksi hyvin paljon Javaa muistuttavassa C#-kielessä aksessorit on toteutettu siten, että ohjelmoijan ei tarvitse kirjoittaa erillisiä metodeita yhden arvon käytön hallintaan. C# ratkaisee ongelman käyttämällä Property-nimisiä kenttiä luokan sisällä. Property on ikään kuin kenttä, mutta sen lukemiselle ja kirjoittamiselle on mahdollista asettaa tämän suorittavat metodit. Tällöin propertyä voi kutsua mukavammalla syntaksilla device.Name tai device.Name = "uusi nimi".

Java-kielessä aksessorien paljouden ongelmaa auttaa jonkin verran kehittyneet tekstieditorit, mutta kuten C#-kielessä, ongelman voisi myös ratkaista kielen tasolla lisäämällä siihen vastaavan ominaisuuden.

8.4 Tyyppijärjestelmä

Javan tyyppijärjestelmästä oli paljon hyötyä koodin kehityksessä. Kuten kaikissa vahvasti tyypitetyissä ohjelmointikielissä, kääntäjä kertoo ohjelmoijalle, kun metodeihin syötetään väärän tyyppistä dataa. Lisäksi kääntäjä antaa varoituksen, kun metodit yrittävät palauttaa väärän tyyppistä tietoa. Java ei tarjoa tyyppijärjestelmänsä puolesta muihin kieliin verrattuna mitään uutta, hyödyllistä toiminnallisuutta ohjelmoijalle.

8.5 Poikkeukset

Javan poikkeuksienkäsittely on toteutettu hyvin. Java-kieli pakottaa ohjelmoijan merkitsemään kaikki metodit, jotka heittävät poikkeuksia. Metodeihin yksinkertaisesti kirjoitetaan heitettävien poikkeuksien tyypit. Tämä menettely teki koodin lukemisesta hyvin helppoa.

9 Javan havaitut puutteet

Java-kielen ominaisuudet, vaikka ovatkin toimivia, ovat kuitenkin kankeita verrattuna vaihtoehtoisiin kieliin. Käsittelen tässä osassa Javan ominaisuuksia, joita koin kehityksen aikana, että Javassa olisi hyvä olla.

9.1 Metaohjelmointi

Metaohjelmointi tarkoittaa ohjelman toiminnallisuuden luontia tai muuttamista ohjelmallisesti. Se olisi antanut valtavasti vapauksia toistuvien luokkien ja metodien kirjoittamiseen. Mielenkiintoinen ja hyödyllinen lisä Javaan olisi metaohjelmointiominaisuuksien lisääminen kieleen.

Metaohjelmoinnin yksi osa, jota Javaan voitaisiin soveltaa, on ohjelmakoodin tuottaminen eli generointi. Siinä missä Java-ohjelmoija joutuu lisäämään kaiken toiminnallisuuden käsin, metaohjelmointi voisi helpottaa ohjelmoijan työtä. Ohjelmoija voisi kirjoittaa toistuvat ohjelman osat vain kerran, ja kutsua kielen tarjoamia metaohjelmointiominaisuuksia niissä kohdin ohjelmaa, missä niitä tarvitaan. Kääntäjä suorittaisi käsketyt metaohjelmointiominaisuuksiin liittyvät käskyt joko koodin kääntämisen tai ajon aikana, ja tuloksena olisi toimiva ohjelma vähemmällä kirjoittamisella.

Vastaava ominaisuus löytyy useista muista kielistä. Esimerkkeinä Ruby, Haskell ja Common Lisp. Javaa paljon muistuttavassa C#-kielessäkin on jonkin verran metaohjelmointiominaisuuksia.

9.2 Ensimmäisen luokan funktiot

Ensimmäisen luokan funktioita olisi voinut käyttää koodimäärän pienentämiseen. Mikäli kieli tukee ensimmäisen luokan funktioita, voidaan luoda funktioita, jotka ottavat parametreikseen toisia funktioita. Tällöin eri tarkoituksiin tehtyjä funktioita on helppo luoda joustavasti. Se, että Javassa ei ole tätä ominaisuutta, teki ohjelmakoodin määrästä jonkin verran suuremman.

9.3 Implisiittiset tyyppijulistukset

Implisiittiset tyyppijulistukset ovat keino pyytää ohjelmakoodin kääntäjää määrittämään kentän tyyppi sen sijaan, että ohjelmoija määrittelisi tyypin käsin kirjoittamalla. Tätä ominaisuutta käytettäessä koodista tulee lyhempää ja usein myös luettavampaa.

Ohjelmoija voi esimerkiksi määrittää monimutkaisen tyypin, vaikkapa merkkijonoja sisältävän taulukon sisältävän listan kirjoittamalla yhden avainsanan kahden sijasta. Javaa muistuttava C#-kieli tukee tätä ominaisuutta avainsanalla "var" (Microsoft 2012). Kentän tyyppin julistuksessa voidaan esimerkiksi kirjoittaa "'var koiranNimi = "musti"', jolloin kentän koiranNimi tyypiksi pääteltäisiin merkkijono (string).

Tämä ominaisuus löytyy luonnollisesti myös dynaamisesti tai heikosti tyypitetyistä kielistä, sillä niissä ei tavallisesti määritellä kentän tyyppiä yhtä tiukasti kuten esimerkiksi Javassa.

9.4 Osittaisluokat

Osittaisluokkien tukeminen olisi hyvä lisä Java-kieleen. Osittaisluokka -ominaisuus tarjoaa ohjelmoijalle mahdollisuuden jakaa yhden konkreettisen luokan määrittely useisiin lähdekoditiedostoihin.

Ominaisuus on hyödyllinen esim. silloin kun ohjelmoija kirjoittaa graafista ohjelmaa, ja hän haluaa eritellä käyttöliittymän toteutuksen sekä sen alla olevan toiminnallisuuden toisistaan. Esimerkiksi C#-kielessä, joka tukee ominaisuutta, käyttöliittymän koodi sijoitetaan omaan osittaisluokkaansa, ja muu luokan toiminnallisuus pidetään omassa osittaisluokassansa.

Java-kielessä ohjelmoija on sen sijaan pakotettu kirjoittamaan yhden luokan määrittely yhteen tiedostoon. Vaikka tämä rajoitus edesauttaakin koodin järjestelyä, ei sen pitäisi olla pakollinen. Ohjelmoija voisi, mikäli Java tukisi ominaisuutta, jäsennellä esimerkiksi yhden luokan vakiot omaan tiedostoonsa. Suurikokoisissa luokissa ominaisuus olisi hyvin tarpeellinen.

9.5 Sivuvaikutuksien merkintä metodien tyyppeihin

Javan tyyppijärjestelmää voisi kehittää siten, että siinä olisi mahdollisuus merkitä metodien aiheuttamat sivuvaikutukset metodien tyyppimerkintöihin. Kun koodin lukija saisi heti metodin tyyppimerkinnästä tietää, muuttaako metodin kutsuminen olioiden tai ulkomaailman tilaa, tulisi koodin lukemisesta helpompaa.

Tämä ominaisuus on toteutettu joissakin muissa kielissä, kuten Haskell-kielessä pakollisena, sekä C#-kielessä vapaaehtoisena. Se on kuitenkin vielä olio-ohjelmoinnin viitekehykselle melko uusi.

9.6 Tiukka olio-viitekehykseen sitominen

Yhtenä Javan ongelmana päättelen olevan sen, että siinä ohjelmoija sidotaan tiukasti olio-viitekehykseen, eikä funktionaalista tai proseduraalista koodia ole mahdollista tai kätevää kirjoittaa.

Tiukasta viitekehyksestä on sekä hyötyä että haittaa. Hyötynä on, että kaikki Java-ohjelmat ovat hyvin yhtenäisiä, joten niiden opiskelu ja jatkokehittäminen onnistuvat helposti useimmilta kokeneilta Java-ohjelmoijilta. Haittana on, että ohjelmien kirjoitus soveltuu parhaiten yhdentyyppisten ongelmien ratkaisuun yhdellä tavalla. Mikäli ohjelmoija haluaa toteuttaa ratkaisun jollain muulla ohjelmointityylillä kuin olio-ohjelmoinnilla, tämä on joko vaikeaa ja hidasta tai kokonaan mahdotonta.

Monet muut modernit, olio-ohjelmointia tukevat ohjelmointikielet tukevatkin useita ohjelmoinnin viitekehyksiä samanaikaisesti. Tällaisia kieliä ovat C#, Python, Ruby, Perl ja jopa PHP. Viitekehyksen laajentaminen voisi olla Javalle hyödyllinen lisä, sillä useita viitekehyksiä tukeva kieli on väistämättä voimakkaampi ilmaisuväline ohjelmoijalle. Vaikka uusien ominaisuuksien lisääminen aiheuttaakin monille ohjelmoijille uuden opettelun tarvetta, saadaan sillä se hyöty, että joitakin ohjelmointirakenteita on huomattavasti helpompi käsitellä.

Javaan itse asiassa ollaankin suunnitelemassa viitekehyksen laajentamista. Joitakin funktionaalisen ohjelmoinnin ominaisuuksia, kuten lambda-lausekkeita, ollaan lisäämässä Java-kielen tuleviin versioihin. (Goetz ym. 2011.)
Itse JVM-virtuaalikone pystyy kyllä suorittamaan funktionaalista koodia loistavasti. Tästä esimerkkeinä ovat sen päällä suoritettavat funktionaaliset kielet Clojure ja Scala (École Polytechnique Fédérale de Lausanne 2012; Hickey 2012).

10 Java graafisen käyttöliittymän kehityksessä

Käyttöliittymän kehitys onnistui melko vaivattomasti. Java tarjoaa käyttöliittymän kehitykseen hyvän graafisen kirjaston nimeltä Swing. Swing tarjoaa ohjelmoijalle muihin käyttöliittymäkirjastoihin verrattuna tavalliset kehitysominaisuudet, joten kokenut ohjelmoija pääsee sillä nopeasti alkuun.

Käyttöliittymän suunnittelu toteutettiin kahdessa osassa. Ensimmäinen osa oli käyttöliittymän prototyyppi, ja toinen varsinainen toteutus. Alkuperäinen prototyyppi oli toteutettu ilman graafista käyttöliittymän suunnittelutyökalua, jälkimmäisen suunnittelussa puolestaan käytettiin NetBeans-nimistä ohjelmointiympäristöä.

Javan vahva tyypitys oli käyttöliittymän kehityksessä enemmän hidasteena kuin hyötynä. Kaikki ohjelmassa esiintyvät kentät esiintyvät kentät luettiin DSiP-asetustiedostoista sisään merkkijonoina.

Ongelmaksi muodostuikin arvojen serialisointi, eli niiden muunto tallennukseen kelpaavasta muodosta oliomuotoon. Kaikki kentät oli tallennettu merkkijonoina DSiP-asetustiedostoihin, ja niistä tuli muuntaa helposti käsiteltäviä olioita. Esimerkiksi joissakin luokissa olevat kentät määriteltiin kokonaisluku-tyyppisiksi (Integer). Tästä seuraa se, että kenttiin ei voi sijoittaa esim. merkkijonoa (String), vaan kokonaisluku pitää muuntaa merkkijonoksi ennen kuin sen voi sijoittaa muuttujaan. Tähän on käytettävä ylimääräistä muuntometodia, esim. Integer-luokan staattista toString()-metodia: Integer.toString(muuttuja);.

Kenttien vahva tyypitys aiheutti täten muilla kuin merkkijono-tyyppisillä kentillä sen, että ennen niihin kirjoittamista piti jälleen suorittaa tyyppimuunnos. Koska tämä oli toistuva operaatio, oli järkevää määrittää omat metodinsa yleisille tyyppimuunnoksille. Metodien määrä nousi yllättävän korkeaksi, sillä useimmille kentille luotiin erikseen natiivityypin sekä merkkijonotyypin aksessorit.

Koska kenttien määrä lähentelee neljääkymmentä, käytin useimpien aksessorien kirjoitukseen eräänlaisia tekstieditorilla toteutettuja koodigeneraattoreita. Vaikka ratkaisu toimii, suuri määrä aksessoreita on vaikeasti ylläpidettävä ja ratkaisua on hyvin kankea muuttaa jälkikäteen. Hyvä vaihtoehto erillisten aksessorien tekemiseen olisi ollut luoda oma luokka samantyyppisille kentille. Tällöin kaikkien samantyyppisten kenttien aksessorit olisi tarvinnut kirjoittaa vain kerran, sillä ne oltaisiin hallittu luokan sisällä.

Vaihtoehtona olisi voinut olla käyttää jotakin metaohjelmointitekniikkaa, jolla kaikki aksessorit luotaisiin juuri ennen ohjelman kääntämistä. Tähän en kuitenkaan työn edetessä halunnut ryhtyä, vaan halusin rajata projektin teeman Java-kielen ominaisuuksien tutkimiseen.

Javan kirjastot tarjoavat metodeja juuri tällaisiin tyyppimuunnoksiin, mutta ohjelman tarkoitukselle sopi paremmin mukautettujen metodien luonti, sillä tällöin voitiin varmistua vähemmällä poikkeuksienkäsittelyllä, että mikäli muunnos epäonnistuisi, muuttujaan sijoitettaisiin silti oletusarvo.

11 Javan yhteensopivuus Gitin kanssa

Git-versionhallinta sopi ohjelman kirjoittamiseen oikein hyvin. Versionhallinta osoittautui erittäin hyväksi työkaluksi koodin hallintaan. Työn edetessä siitä oli hyötyä koodin jakamiseen eri kehityskoneiden välillä, eri ominaisuuksien samanaikaisessa kehityksessä sekä virheiden korjauksessa palauttamalla kehityksessä mukana olleiden tiedostojen vanhoja versioita.

Java soveltui tutkimuksen aikana versioitavaksi yhtä hyvin kuten muutkin yleiset ohjelmointikielet. Kaikki sen lähdekoodi oli selväkielistä, minkä vuoksi Git osasi tulkita kooditiedostoja helposti. Vaikka kaikkea koodia ei kirjoitettu käsin, vaan osa siitä tuotettiin käyttöliittymäeditorilla, binäärimuotoista koodia ei ollut tarvetta versioida missään vaiheessa.

Kehityksen yhteydessä käytössä oli versionhallinnan osalta neljä työpistettä sekä kaksi keskitettyä Git-palvelinta. Koodin synkronointi ja jakaminen kaikkien näiden tietokoneiden välillä sujui Gitin avulla vaivatta. Git-järjestelmä havaittiin taipuisaksi eri käyttötarkoituksiin sekä hyvin vikasietoiseksi. Lisäksi Javan monialustaisuuden ansiosta kehitys eri käyttöjärjestelmien välillä sujui täysin saumattomasti.

12 Vaihtoehtoisia ohjelmointikieliä Javalle

Aiemmin esittelemäni puutteet Java-kielessä voivat tuntua pieniltä tai merkityksettömiltä, mikäli lukija ei ole tutustunut useisiin eri ohjelmointikieliin. Paul Grahamin mukaan onkin vaikeaa ymmärtää jonkin ohjelmointikielen puutteita ennen kuin on kokeillut ilmaisuvoimaltaan voimakkaampia kieliä. Tämänkaltaisessa tilanteessa oleva henkilö näkee Grahamin mukaan voimakkaammat kielet "outoina", ensinnäkin koska ei ole kokenut niiden ilmaisuvoimaa, ja toisekseen, koska on omaksunut tutulle kielelleen ominaisen ajattelutavan. (Graham 2001.)

Toisaalta henkilö, joka on ohjelmoinut ilmaisuvoimaltaan voimakkaammalla ohjelmointikielellä, näkee heikomman kielen puutteet terävästi. Grahamin mukaan mikäli heikommalla ohjelmointikielellä ohjelmointiin ei ole syytä, on heikomman kielen valinta virhe. (Graham 2001.)

Hänen mukaansa kaikki ohjelmointikielet eivät ole tasavertaisia, vaan ne voidaan ensinnäkin sijoittaa jatkumolle sen tason mukaan, jolla ne antavat ohjelmoijan ohjelmoida. Toisekseen itse kielet vaihtelevat ilmaisuvoimiensa suhteen. (Graham 2001.)

Samaa ajatusta tukee myös Douglas Crockford, yksi maailman suosituimman ohjelmointikielen, JavaScript-kielen, suunnittelijoista. JavaScript on Internet-selaimissa suoritettava ohjelmointikieli, ja koska Internet on hyvin yleinen, on JavaScriptistä tullut hyvin levinnyt ja suosittu. (Crockford 2001.)

Crockford huomauttaa, että Javaa jatkokehitettiin alun perin, sen jälkeen kun kieltä oltiin kehitetty StarSeven-laitteelle, Internet-selaimissa suoritettavaksi kieleksi. JavaScript kuitenkin syrjäytti Javan Internet-selainten kielenä, koska se on Crockfordin mukaan parempi kieli. (Crockford 2011.)

Päättelen, että mikäli edellä mainitut syyt Java-kielen valinnaksi projektin kielenä eivät täyty, ei ole painavaa syytä käyttää Javaa. Vaihtoehtoisina kielinä voidaan käyttää vaikkapa Javaa hyvin paljon muistuttavaa C#-kieltä tai tyyppiturvallista ja selkeäsyntaksista Haskellia.

Mikäli rajoitteena on resurssien puute, voidaan käyttää esimerkiksi Groovy-kieltä. Tämä kieli rakentuu Javan syntaksin ketteräksi jatkeeksi, ja sen luvataan tarjoavan moderneja ohjelmointiominaisuuksia, joiden opettelu ei vie juuri lainkaan aikaa tottuneille Java-ohjelmoijille. Esimerkkihyöty Groovy-kielestä on implisiittinen tuki aksessorien luonnille, joka muistuttaa hieman C#-kielen Property-ominaisuutta.

Mikäli rajoitteena on JVM-virtuaalikone, voidaan silti käyttää uudenaikaisempia kieliä, joita ajetaan JVM:n päällä. Näistä esimerkkeinä ovat Clojure sekä Scala, joita kumpaakin esittelen lyhyesti seuraavassa osassa. Kumpikin kieli on ominaisuuksiltaan selvästi Javaa kehittyneempi, ja kumpaakin kieltä on mahdollista suorittaa Javan virtuaalikoneella.

12.1 Clojure

Clojure on yksi Lisp-ohjelmointikieliperheen murteista, ja se tukee kehittyneitä metaohjelmoinnin ominaisuuksia. Paul Graham pitää Lisp-kieliä voimakkaimpina olemassa olevina ohjelmointikielinä (Graham 2001).

Clojure tukee monia Java-maailmalle vieraita mutta tehokkaita ominaisuuksia. Se lainaa funktionaalisesta viitekehyksestä muuttumattomat tietorakenteet, joilla voidaan luotettavasti käsitellä tietoa, ilman pelkoa siitä, että jokin metodi muuttaa sitä sivuvaikutuksenaan. Clojure ei käytä olio-viitekehystä, vaan on funktionaalinen. Se tukee Lisp-kieliperheen voimakkaita ominaisuuksia, kuten "koodi suoritettavana tietorakenteena"-ominaisuutta ja makroja.

Clojure tukee tyyppipäättely-ominaisuutta. Tällä ominaisuudella tarkoitetaan kääntäjän kykyä päätellä jonkin metodin odotettu paluutyyppi muiden metodien parametrien tyypeistä. Tällöin ohjelmoijan ei tarvitse erikseen kirjoittaa ohjeita tyyppien käsittelystä. Esimerkiksi kokonaislukuja käsittelevälle funktiolle ei tarvitse erikseen määrittää paluutyypiksi kokonaislukua, vaan tyyppipäättelyä tukeva kääntäjä voi päätellä sen funktion toteutuksesta.

Sitä voidaan lisäksi kääntää kirjoitushetkellä JVM-virtuaalikoneelle, C#-kielen virtuaalikoneelle (CLR-virtuaalikoneelle) sekä JavaScript-kieleksi. Clojuren luvataan helpottavan monisäikeisten, eli usealla ytimellä samanaikaisesti suoritettavien ohjelmien kirjoittamista. (Hickey 2012.)

12.2 Scala

Scala-kieli puolestaan on JVM:llä ajettava funktionaalinen oliokieli. Sen ensimmäinen versio on julkaistu vuonna 2001, ja on nykyään hyvä vaihtoehto Java-kielelle. Scalaa käyttää mm. Twitter-verkkopalvelu viestinvälityksensä ytimessä. Scala toimii hyvin yhteiskäytössä Javan kanssa, ja sen käytön luvataan tuottavan toimivan koodin puolella tai kolmanneksella siitä rivimäärästä, mitä koodissa olisi Javalla kirjoitettuna. Scala täyttää siis aiemmin mainitun moniparadigmakielen määritelmän hyvin. (École Polytechnique Fédérale de Lausanne 2012.)

13 Yhteenveto Java-kielen ominaisuuksista ohjelman kehityksessä

Tutkimuksen tavoitteena oli selvittää Java-ohjelmointikielen ohjelmoijalle tarjoamia ominaisuuksia. Tutkimus suoritettiin rakentamalla vikasietoisen DSiP-reititysjärjestelmän konfiguraatioeditoriohjelma. Java-kieli soveltui pääosin hyvin ohjelman kirjoittamiseen. Se tarjoaa hyvät ja yleisessä käytössä olevat perustyökalut ohjelmien tuotantoon. Javan virtuaalikone JVM suoriutui ohjelman suorituksesta loistavasti eri alustoilla testattaessa. Ohjelman lopputulos oli toimiva ja mukava käyttää.

Itse kehitysprosessi ei kuitenkaan ollut niin joustava kuin mitä se olisi voinut olla muilla ohjelmointikielillä kirjoitettaessa. Tutkimuksen aikana Java havaittiin melko rajoittuneeksi kieleksi. Tutkimuksen yhteydessä Javaa vertailtiin toisiin moderneihin ohjelmointikieliin, ja havaittiin Javasta puuttuvan monia tärkeitä ominaisuuksia. Nämä puutteet tulkittiin tekevän ohjelmoijan työstä vaikeampaa ja monimutkaisempaa.

Java osoittautui tutkimuksen aikana hyväksi mutta rajoittuneeksi ohjelmointikieleksi. Vaikuttaa siltä, että Javaa kannattaa käyttää vain, mikäli tilanne ennen kehityksen aloittamista täyttää jommankumman seuraavista ehdoista: joko ohjelma täytyy integroida jonkin olemassa olevan Java-ohjelman tai -ympäristön kanssa, tai kehittäjät eivät osaa vaihtoehtoisia kieliä, eikä heillä ole resursseja tai motivaatiota opetella niitä.

Voitaisiin myös sanoa, että yksi syy käyttää Java-kieltä on se, että koodia voidaan ajaa JVM-virtualikoneella, ja kehityksessä voidaan käyttää Javan valtavia ohjelmakirjastoja. Tämä ei kuitenkaan itsessään voi riittää perusteluksi, sillä JVM-virtuaalikonetta sekä Javan kirjastoja voidaan hyödyntää toisilla, paremmilla kielillä kuin Java.

Jatkokysymykseksi Java-kielen käytöstä jääköön se, millä tavalla Java-kieltä käyttänyt organisaatio voi tehokkaimmin kehittää ohjelmistokehitysprosessiaan siirtymällä käyttämään kehittyneempiä ominaisuuksia tarjoavia ohjelmointikieliä. Tutkimuksessa on osoitettu jopa Javan ympäristössä toimivia kehittyneempiä kieliä, joihin siirtyminen yritysmaailmassa saattaisi olla kannattavaa.

Lähteet

Kirjalliset

Holmström, Rajamäki ja Hult. The future solutions and technologies of public safety communications - DSiP traffic engineering solution for secure multichannel communication. International Journal of Communications . 115-117.

Sähköiset

Oracle 2011. The Java Tutorials. Viitattu 14.11.2011.
http://java.sun.com/docs/books/jls/second_edition/html/intro.doc.html

Oracle 2011. The Java Tutorials. Viitattu 18.11.2011.
http://java.sun.com/docs/books/jls/second_edition/html/typesValues.doc.html

Microsoft 2012. C# Reference. Viitattu 14.1.2012.
http://msdn.microsoft.com/en-us/library/bb383973.aspx

Caprio 2006. Virtual Machines: Virtualization vs. Emulation. Viitattu 26.1.2012.
http://www.griffincaprio.com/blog/2006/08/virtual-machines-virtualization-vs-emulation.html

Salonen 2011. Javasta C#:iin. Viitattu 18.2.2012.
http://villesalonen.fi/2011/javasta-ciin/

Sun Microsystems Inc 2001. Java Platform Performance: Strategies and Tactics. Appendix A:The Truth About Garbage Collection.
http://java.sun.com/docs/books/performance/1st_edition/html/JPAppGC.fm.html

Goetz et al. 2011. JSR-000335 Lambda Expressions for the Java Programming Language - Early Draft Review
http://jcp.org/aboutJava/communityprocess/edr/jsr335/index.html

École Polytechnique Fédérale de Lausanne 2012. The Scala Programming Language. Viitattu 19.2.2012.
http://www.scala-lang.org/node/25

Hickey 2012. Clojure – rationale. Viitattu 19.2.2012.
http://clojure.org/rationale

SpringSource 2012. Groovy – Home. Viitattu: 19.2.2012.
http://groovy.codehaus.org/

Microsoft 2012. Functional Programming vs. Imperative Programming. Viitattu: 14.1.2012.
http://msdn.microsoft.com/en-us/library/bb669144.aspx

Graham 2001. Beating the Averages. Viitattu: 19.2.2012.
http://www.paulgraham.com/avg.html

Crockford 2011. Crockford on JavaScript: A Public Lecture Series at Yahoo!. Viitattu: 19.2-.2012.
http://yuiblog.com/crockford/

Crockford 2001. JavaScript: The World's Most Misunderstood Programming Language. Viitattu: 19.2.2012.
http://www.crockford.com/javascript/javascript.html

Ajeco 2012. The DSiP-solution with multichannel routing capability. Viitattu: 14.1.2012. http://ajeco.fi/product1.html

Kuvat